



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 17/60	A1	(11) International Publication Number: WO 99/04354 (43) International Publication Date: 28 January 1999 (28.01.99)
---	----	---

(21) International Application Number: PCT/JP97/02460

(22) International Filing Date: 15 July 1997 (15.07.97)

(71) Applicants (for all designated States except US): SHINKO ELECTRIC INDUSTRIES CO., LTD. [JP/JP]; 711, Aza Shariden, Oaza Kurita, Nagano-shi, Nagano 380 (JP). SILICON AUTOMATION SYSTEMS PVT. LTD. [IN/IN]; 3008, 12th "B" Main, 8th Cross, Hal 2nd Stage, Indiranagar, Bangalore-560 008 (IN).

(72) Inventors; and

(75) Inventors/Applicants (for US only): SEKIGAWA, Kazunari [JP/JP]; Shinko Electric Industries Co., Ltd., 711, Aza Shariden, Oaza Kurita, Nagano-shi, Nagano 380 (JP). SATO, Tomomi [JP/JP]; Shinko Electric Industries Co., Ltd., 711, Aza Shariden, Oaza Kurita, Nagano-shi, Nagano 380 (JP). PATEL, Supriya, K. [IN/IN]; Silicon Automation Systems Pvt. Ltd., 3008, 12th "B" Main, 8th Cross, Hal 2nd Stage, Indiranagar, Bangalore-560 008 (IN). SANKOLLI, Sanjay, D. [IN/IN]; Silicon Automation Systems Pvt. Ltd., 3008, 12th "B" Main, 8th Cross, Hal 2nd Stage, Indiranagar, Bangalore-560 008 (IN).

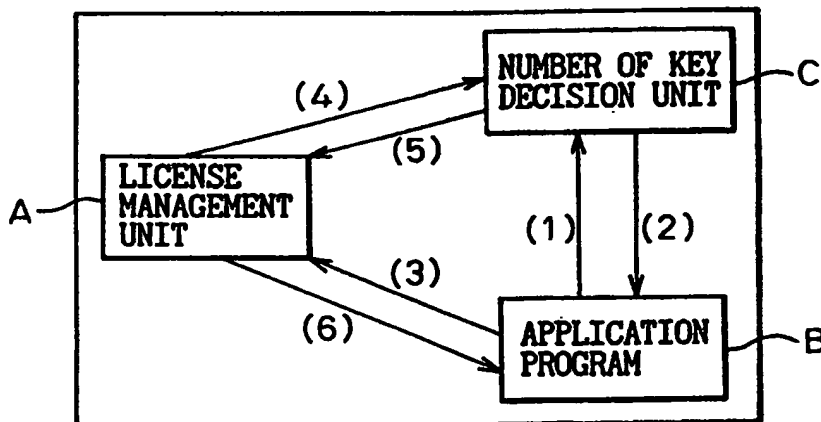
(74) Agents: ISHIDA, Takashi et al.; A. Aoki & Associates, Toranomom 37 Mori Building, 5-1, Toranomom 3-chome, Minato-ku, Tokyo 105 (JP).

(81) Designated States: JP, KR, US.

Published

With international search report.

(54) Title: A LICENSE MANAGEMENT SYSTEM



(57) Abstract

A license management system for software which drives a single computer or a plurality of computers includes: an application program for requesting a decision of the number of license which it needs to drive itself and for receiving issuance of the license; a number of license decision unit for determining the necessary number of licenses in accordance with the request from the application program; and a license management unit for issuing the number of licenses which was determined by the number of license decision unit. Further, the number of license decision unit has means for determining the number of licenses based on the following multi-nominal function, $LK = f(x_1, x_2, \dots, x_n)$, where, LK is the number of licences, and x_1 to x_n are parameters which are needed to determine the number of licenses. According to the present invention, it is possible to provide a license management system enabling issuance of a license in which the sales strategy of a software maker was considered.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

- 1 -

DESCRIPTION

A LICENSE MANAGEMENT SYSTEM

TECHNICAL FIELD

5 The present invention relates to a license management system for software, particularly, it relates to a license management system which can flexibly issue a license for software to a user in accordance with an environment in use of the user when a software maker
10 supplies the desired software to the user.

BACKGROUND ART

 Recently, computers have been widely utilized in various fields, and not only development of the software itself, but also license management for the software is
15 important when the software maker issues the software to the user or after issuance of the software.

 In general, the software have been developed by the software maker, and have been used by the user who has a hardware (computer) based on a predetermined license
20 contract. After the software was supplied to the user, the software maker performs the license management, i.e., checks as to whether the software is used properly based on the predetermined license contract.

 Conventionally, there are mainly two types of
25 license contracts, i.e., a floating-license system and a node-LOCK license system. In the former, the number of the licenses are decreased one by one for every start of the software, and are increased one by one for every termination thereof. In the latter, the hardware to be
30 used is fixed in the user side, and the software maker issues the license at a predetermined constant price regardless the number of the software.

 Further, in the former, when the user wants to operate a plurality of workstations (for example, 10) by
35 using the same software, the software maker takes the license contract from the user in the price of ten licenses.

- 2 -

According to the former, the software maker has an advantage in which he can set a high unit price of the software. Furthermore, the user also has an advantage of flexibility in use of the software since he can optionally change the number of the hardware to be used.

On the other hand, in the latter, the hardware to be used in the user is fixed, and the software maker issues the license contract in a predetermined constant price regardless the number of the software and performance of the software.

According to the latter, the user has an advantage in which he can set a low unit price of the software. However, since the hardware to be used is fixed, there is no flexibility in use of the software. Further, since all the software is handled by only one hardware, there is a problem in which the whole throughput of the system becomes worse.

As explained above, there are various problems for both software maker and the user in the conventional license system. Particularly, there is a problem in which the software maker cannot issue the license which the sales strategy was considered. Accordingly, the software maker has always searched for and developed an optimum license management in order to supply the software, which have been developed at large expense, to the user, at a suitable price.

DISCLOSURE OF THE INVENTION

The object of the present invention is to provide a license management system enabling issuance of a license in which the sales strategy of the software maker was considered.

The present invention is a license management system for software which drives a single computer or a plurality of computers, and includes an application program for requesting a decision of the number of the license which needed to drive itself and for receiving issuance of the license, a number of license decision

- 3 -

unit for determining the necessary number of license in accordance with the request from the application program, and a license management unit for issuing the number of license which was determined by the number of license decision unit, to the application program.

The number of license decision unit includes means for determining the number of licenses based on the following multi-nominal function.

$$LK = f(x_1, x_2, \dots, x_n)$$

Where, LK is the number of licenses, and x_n is a parameter which is needed to determine the number of license.

Further, the present invention is a license managing method in a license management system including at least a license management unit, an application program, and a number of license decision unit, and for driving a single computer or a plurality of computers, the method comprising the steps of;

delivering parameters from the application program to the number of license decision unit in order to substitute the number of license, which needed to start the application program, for the multi-nominal function, when the application program starts;

determining the number of license by checking values of the necessary parameters which need to determine the number of licenses in the number of license decision unit, substituting the determined number of license for the parameters delivered from the application program, and returning the parameters to the application program;

notifying the necessary number of license from the application program to the license management unit by delivering the parameters, and requesting the license; and

subtracting the necessary number of license from the number of licenses which are held in the license management unit when the license management unit receives a normal flag from the number of license decision unit,

- 4 -

and returning the normal flag to the application program when the number of licenses, which are held in the license management unit, is not negative, and issuing the license to the application program.

5 BRIEF EXPLANATION OF DRAWINGS

Figures 1 to 5 are explanatory views of sales strategy for a software maker;

10 Figure 6 is a basic structural view of a first embodiment of a license management system according to the present invention;

Figure 7 is a basic structural view of a second embodiment of a license management system according to the present invention;

15 Figure 8 is a basic structural view of a third embodiment of a license management system according to the present invention;

Figure 9 is a basic structural view of a fourth embodiment of a license management system according to the present invention;

20 Figure 10 is an explanatory view of a start process between a license daemon and an application program;

Figure 11 is an explanatory view of a fork process in the license daemon;

25 Figure 12 is an explanatory view of the fork process in the application program;

Figure 13 is an explanatory view of a termination process of the application program;

Figure 14 is an explanatory view of a compulsory termination from a private application manager;

30 Figure 15 is an explanatory view of a termination report of the application program;

Figure 16 is an explanatory view of the termination process to the private license daemon;

35 Figure 17 is an explanatory view of a check request from the private license daemon;

Figure 18 is an explanatory view of a normal notification from the private application manager;

- 5 -

Figure 19 is an explanatory view of a restart process from the private license daemon;

Figure 20 is an explanatory view of a license releasing process in the private license daemon;

5 Figure 21 is an explanatory view of the check request from the private application manager to the private license daemon;

10 Figure 22 is an explanatory view of the normal notification from the private license daemon to the private application manager;

Figure 23 is an explanatory view of invalid information in the private application manager;

Figure 24 is an explanatory view of the termination process in the private application manager;

15 Figure 25 is an explanatory view of a restart in the private license daemon;

Figure 26 is an explanatory view of an invalid notification from the license daemon;

20 Figure 27 is an explanatory view of a restart notification from the license daemon;

Figure 28 is an explanatory view of a polling process from the private license daemon to the license daemon;

25 Figure 29 is an explanatory view of the termination process in the license daemon;

Figure 30 is an explanatory view of the termination process in the license daemon, when the private application manager APCM1 does not receive the instruction COR from the application program AP1 within a predetermined time in the process;

30 Figure 31 is an explanatory view of an abnormal termination in the license daemon;

Figure 32 is an explanatory view of the abnormal termination in the application program;

35 Figure 33 is an explanatory view of the abnormal termination between the license daemon and the private application manager;

- 6 -

Figure 34 is an explanatory view of the abnormal termination in both the application program and the private application manager;

5 Figure 35 is an explanatory view of the abnormal termination in both the application program and the private license daemon;

Figure 36 is an explanatory view of the abnormal termination in both the license daemon and the private license daemon;

10 Figure 37 is an explanatory view when the license daemon LDP restarts after the situation described in the explanation of Fig. 36;

Figure 38 is an explanatory view of a system structure;

15 Figure 39 is a flowchart for determining the number of license (the number of key);

Figures 40(A) to 40(I) are actual function lists for determining the number of key;

20 Figures 41(A) to 41(F) are explanatory views of one example of the program; and

Figures 42(A) to 42(C) are explanatory views of a result of execution of the same program.

BEST MODE OF CARRYING OUT THE INVENTION

25 Figures 1 to 5 are explanatory views of sales strategy in a software maker. In this specification, a license is called a "key", and the number of licenses is called "the number of keys". In the present license system, the number of keys is equal to the number of licenses, and it is possible to consider the number of
30 keys as the same as the value of the software. However, in order to realize suitable license management, it is desirable to realize a predetermined relative relationship between the number of licenses and the number of keys (this relationship corresponds to a
35 function "f" as mentioned below). For example, it is desirable to have the relationship in which the number of keys is proportional to the number of licenses.

- 7 -

The following five patterns must be considered in the sales strategy of the software maker as shown in Figs. 1 to 5.

5 As shown in Fig. 1, when the application program is particularly designed so as to operate at high speed in accordance with a high speed hardware including many memories, the design of the software becomes complicated so that the software has a high value. Accordingly, when the hardware having many memories is used in a user side, 10 the software maker should request the more number of keys to the user in accordance with the more number of memories.

As shown in Fig. 2, in the case of Fig. 1, when a performance of the computer is high, the throughput of 15 the computer per a unit time is also high. Accordingly, the software maker should request the more number of keys to the user in accordance with the higher performance of the computer.

As shown in Fig. 3, in the case that the application 20 program is not particularly designed, according to a view point of the user, since the performance of the computer becomes higher so that an appearance performance of the software also becomes higher, the application program should be operated based on less number of keys. 25 According to this view point, the software maker should consider so as to be able to start the software based on less number of keys. However, the above is particular case. For example, when the software is on the way of development and design, and when the performance thereof is not finally determined in the software maker, the 30 software maker wishes to use that software in the user side in order to improve the performance.

As shown in Fig. 4, for the user who buys a large amount of software, the necessary number of keys are 35 gradually reduced in accordance with the number of purchased the keys. Accordingly, in this case, it is necessary to reduce a substantial unit price of the

- 8 -

software in the software maker.

As shown in Fig. 5, the software maker supplies the software free of charge until a predetermined date (this corresponds to "the necessary number of keys is zero"), and supplies the software for a fee after that predetermined date (this corresponds to "the number of key is larger than zero").

As explained above, it is necessary to provide the license management system so as to determine the number of key in accordance with various factors in the use of the software, i.e., a factor caused by the hardware, a factor caused by the processing time, a factor caused by cost, etc..

The present invention aims to perform issuance of the number of keys enabling setting of parameters in flexibility in the software maker.

In order to solve the above problem, the present invention comprises a function for determining the necessary number of keys (the number of key decision unit). The number of license decision unit determines the number of licenses based on the following multi-nominal function.

$$LK = f(x_1, x_2, \dots, x_n) \quad \dots (1)$$

Where, LK is the number of licenses, and x_n is a parameter which needs to determine the number of licenses. As parameters, as explained in Figs. 1 to 5, there are the memory capacity of the hardware, the clock speed of the system, and items which can set freely by the software maker and so on.

Further, the present invention comprises a function for acquiring the values of parameters which are needed to determine the number of keys (parameter value acquiring means).

When the number of key LK is set so as to always become "1", this becomes equivalent to the conventional system. This means that the system of the present invention is ranked to an upper position of the

- 9 -

conventional system, and is compatible with the conventional system. Accordingly, based on the function of the present invention, it is possible to realize an issuance of the number of keys in which the sales
5 strategy was considered in the software maker. Further, it is possible to utilize the fixed number of key in the conventional art. As a result, it is possible to provide an improved license management system for the software.

10 Figure 6 is a basic structural view of the first embodiment of the license management system according to the present invention. In the drawing, A denotes a license management unit, B denotes an application program, and C denotes a number of key decision unit (daemon program). As is known, a daemon program is the
15 program which is automatically executed in a background based on an operating system (OS). In the present invention, all transmission and reception of the data are performed through a network UNIX systems using, for example, the TCP/IP (Transmission Control
20 Protocol/Internet Protocol). In the following drawings, each numeral denotes an order of process.

Process 1: When the application program B starts, it delivers parameters (for example, NoOfKeys) to the number of license decision unit in order to substitute
25 necessary number of keys which need to start the application program.

Process 2: The number of key decision unit C checks values of the necessary parameters which are needed to determine the number of keys and determines the number of
30 keys, substitutes the determined number of keys for the parameters (NoOfKeys) delivered from the application program, and returns the parameters to the application program B.

Process 3: The application program B notifies the
35 necessary number of keys to the license management unit A by delivering the parameters (NoOfKeys), and requests the license.

- 10 -

Process 4: The license management unit A notifies the number of keys (NoOfKeys), which are notified from the application program, to the number of key decision unit C in order to confirm whether the number of keys requested by the application program B is correct.

Process 5: The number of key decision unit C compares the number of keys, which is notified from the application program to the license management unit A, with the necessary number of keys which notified to the application program. When the number of keys is correct, the number of key decision unit C returns a normal flag to the license management unit A.

The processes 4 and 5 are optional because these steps are used for confirmation as to whether correct request is performed.

Process 6: When the license management unit A receives the normal flag from the number of key decision unit C, the license management unit subtracts the necessary number of keys from the number of keys which are held therein, returns the normal flag to the application program B when the number of keys which are held in the license management unit, is not negative, and issues the license to the application program B. When the application program B receives issuance of the license, it becomes an executable application program. If the abnormal flag, which indicates rejection, for example, the flag indicating that the number of keys become negative, is returned from the license management unit A, the application program is terminated at that time.

Figure 7 is a basic structural view of the second embodiment of the license management system according to the present invention. In the above first embodiment, the number of key decision unit C operates as an independent daemon program. In this embodiment, the number of key decision function C' is included within the application program as a function call. In this case, it

- 11 -

is different from the first embodiment, the process 1 is applied as the function call of a variable argument, and the process 2 is applied as a return value thereof.

Further, the number of keys is determined by the application program including the number of key decision unit C itself based on the formula (1).

Process 1: When the application program B starts, it delivers the parameter (NoOfKeys) to the number of key decision function C' in order to receive the number of key which it needs to start itself.

Process 2: The number of key decision function C' checks the necessary value of the parameter, and determines the number of keys. Further, the number of key decision function C' substitutes the values for the parameters delivered from the application program B, and returns the parameter (NoOfKeys) with the values to the application program B.

Process 3: The application program B notifies the necessary number of keys to the license management unit A by delivering the parameter (NoOfKeys), and requests the license.

The processes 4 and 5 in the first embodiment are omitted because these steps are optional.

The process 6: When the license management unit A receives the request from the application program B, it subtracts the necessary number of keys from the number of keys which are held in the license management unit A itself. As a result of subtraction, if the stored number of keys is not negative, the license management unit A returns the normal flag to the application program B, and issues the license thereto. When the application program B receives the issuance of the license, it becomes the executable application program. If the abnormal flag is returned from the license management unit A, the application program B terminates at that time.

Figure 8 is a basic structural view of the third

- 12 -

embodiment of the license management system according to the present invention. In the above first embodiment, when changing the multi-nominal function according to the situation at the software maker, a method of directly
5 changing the number of key decision unit C is employed to change multi-nominal function. In the present embodiment, a database D for determining the multi-nominal function is separately provided in order to read the multi-nominal function to the number of key decision
10 unit C. In this case, when changing the multi-nominal function, the database D is changed so that it is not necessary to change the multi-nominal function in the number of key decision unit C.

Process 1: When the application program B starts,
15 it delivers the parameter (NoOfKeys) to the number of key decision function C in order to substitute the number of keys which it needs to start itself.

Process 2: The number of key decision unit C reads the data from the database D (see process 7), and
20 determines the multi-nominal function. Further, the number of key decision function C checks the necessary value of the parameter, and determines the number of key. Still further, the number of key decision function C substitutes the values for the parameters delivered from
25 the application program B, and returns the parameters with the values to the application program B.

Process 3: The application program B notifies the necessary number of key to the license management unit A by delivering the parameter (NoOfKeys), and requests the
30 license.

The processes 4 and 5 in the first embodiment are omitted because these steps are optional.

The process 6: When the license management unit A receives the normal flag from the number of key decision
35 function C, it subtracts the necessary number of keys from the number of keys which are held in the license management unit A itself. As a result of subtraction,

- 13 -

if the stored number of keys is not negative, the license management unit A returns the normal flag to the application program B, and issues the license thereto. When the application program B receives the issuance of the license, it becomes the executable application program. If the abnormal flag is returned from the license management unit A, the application program B terminates at that time.

Figure 9 is a basic structural view of the fourth embodiment of the license management system according to the present invention. In the above second embodiment, the number of key decision function C' is included within the application program B as the function call. In the present embodiment, a database D' for determining the multi-nominal function is separately provided in order to read the multi-nominal function to the number of key decision unit C'. In this case, when changing the multi-nominal function, the database D' is changed so that it is not necessary to change the multi-nominal function in the number of key decision function C'.

Process 1: When the application program B starts, it delivers the parameter (NoOfKeys) to the number of key decision function C' in order to receive the number of key which it needs to start itself.

Process 2: The number of key decision unit C' reads the data from the database D' (see process 7), and determines the multi-nominal function. Further, the number of key decision function C' checks the necessary value of the parameter, and determines the number of keys. Still further, the number of key decision function C' substitutes the values for the parameters delivered from the application program B, and returns the parameters with the values to the application program B.

Process 3: The application program B notifies the necessary number of key to the license management unit A by delivering the parameter (NoOfKeys), and requests the license.

- 14 -

The processes 4 and 5 in the first embodiment are omitted because these steps are optional.

5 The process 6: When the license management unit A receives the request from the application program B, it subtracts the necessary number of keys from the number of keys which are held in the license management unit A itself. As a result of subtraction, if the stored number of key is not negative, the license management unit A returns the normal flag to the application program B, and
10 issues the license thereto. When the application program B receives the issuance of the license, it becomes the executable application program. If the abnormal flag is returned from the license management unit A, the application program B terminates at that
15 time.

The processes shown in Figs. 6 to 9 are explained in detail with reference to the drawings. The following explanations are given in detail to the processes 3 and 6 between the license management unit A and the application
20 program B.

Conventionally, the license management unit A checked whether the operation of the application program was normal, in such a way that one license daemon (daemon program) communicates with each application program.

25 In this method, however, the license daemon must have many complicated communications with each application program. For example, the license daemon sends a check request for checking a predetermined item to each application program, and each application program
30 returns an answer for the check request to the daemon program. In this case, since the answer is returned simultaneously from each application program to the daemon program, many answers become wait states in the license daemon. This is because many loads are applied
35 to the license daemon. As a result, the processing time for each answer in the license daemon is delayed so that the performance of the software also becomes worse.

- 15 -

Accordingly, the present invention aims to reduce the loads in the license daemon and to eliminate the delay of the processing time in the software (i.e., application program).

5 According to achieve the above purpose, one private license daemon (APSM: Application Program Server Manager) and one private application manager (APCM: Application Program Client Manager) are provided in addition to the license daemon and the application
10 program, and various communications are performed between the private license daemon APSM and the private application manager APCM as explained in detail below.

Figure 10 is an explanatory view of the start process between the license daemon and the application
15 program. When a user performs the start process of the application program, the application program AP1 sends a request CIR (Check-In Request) to the license daemon LDP in order to acquire an approval of execution (i.e., license).

20 Figure 11 is an explanatory view of the fork process in the license daemon. The license daemon LDP generates a fork instruction in order to establish the private license daemon APSM1. The private license daemon APSM1 sends the approval of execution CIRC (Check-In Request Confirmation) to the application program AP1 and issues the license. In this case, the following three states are considered.

a) When the approval of execution CIRC cannot be obtained from the private license daemon APSM1, the
30 application program AP1 terminates the process.

b) When the approval of execution CIRC can be obtained from the private license daemon APSM1, but it includes "invalid" contents, the application program AP1 notifies an invalidation to the user.

35 c) When the approval of execution CIRC can be obtained from the private license daemon APSM1, and it includes "valid" contents, the application program AP1

- 16 -

generates the fork instruction in order to establish the private application manager APCM1.

Figure 12 is an explanatory view of the fork process in the application program. As explained in the above item (c), when the approval of the execution CIRC can be
5 obtained from the private license daemon APSM1, and it includes "valid" contents, the application program AP1 generates the fork instruction in order to establish the private application manager APCM1. After this process,
10 the communications as to the license management are performed between the private license daemon APSM1 and the private application manager APCM1. The application program AP1 starts to execute a proper program itself.

Figure 13 is an explanatory view of the termination process of the application program. When the user
15 performs the termination process of the application program AP1, the application program AP1 sends a request for the termination SIGUSR2 (one kind of signal in UNIX) to the private application manager APCM1, and the
20 application program AP1 is terminated.

Figure 14 is an explanatory view of the compulsory termination from the private application manager. For the case, if the application program does not terminate after sending the request for the termination SIGUSR2
25 accidentally, the private application manager APCM1 sets an invalid information +ve (positive value) into a pipe. The application program AP1 reads the invalid information +ve and terminates itself.

Figure 15 is an explanatory view of the termination report of the application program. When the application
30 program AP1 terminates, the private application manager APCM1 sends the termination report CORC (Check-Out Request Confirmation) indicating the termination of the application program AP1 to the license daemon LDP, and the private application manager APCM1 terminates.
35

Figure 16 is an explanatory view of the termination process of the private license daemon. Finally, the

- 17 -

license daemon LDP writes the information into the database after the application program AP1 terminates, and generates a termination instruction SIGKILL (one kind of signal in UNIX) to the private license daemon APSM1 so as to terminate the private application daemon APSM1 itself.

Next, the polling operation between the private license daemon and the private application manager is explained in detail below. The periodical polling is performed between the private license daemon and the private application manager in order to check whether the normal communication is performed therebetween.

Figure 17 is an explanatory view of the check request from the private license daemon. The private license daemon APSM1 sends the check request APPR (Application Program Poll Request) to the private application manager APCM1 in order to check whether the private application manager APCM1 is operating normally.

Figure 18 is an explanatory view of the normal notification from the private application manager. The private application manager APCM1 sends the normal notification APPC (Application Program Poll Conformation) to the private license daemon APSM1 when the check of contents from the private license daemon APSM1 and the check of contents in the private application manager itself APCM1 are successful.

Figure 19 is an explanatory view of the restart process from the private license daemon. When the private license daemon APSM1 gets an abnormal response from the private application manager APCM1 (Heart beat message exchange fails), the private license daemon APSM1 sends the CORC message to the license daemon LDP and terminates itself.

Figure 20 is an explanatory view of the license releasing process in the private license daemon. The license daemon LDP invalidates the application program AP1, and releases the license key which was

- 18 -

assigned to the application program AP1.

Next, the polling operation from the private application manager APCM1 to the private license daemon APSM1 is explained below.

5 Figure 21 is an explanatory view of the check request from the private application manager to the private license daemon. The private application manager APCM1 sends the check request APRR (Application Program Re-validation Request) to the private license
10 daemon APSM1.

 Figure 22 is an explanatory view of the normal notification from the private license daemon to the private application manager. The private license daemon APSM1 sends the normal notification APRC
15 (Application Program Re-validation Confirmation) to the private application manager APCM1 when the check of the contents from the private application manager APCM1 and the check of the contents in the private license daemon itself are successful.

20 Figure 23 is an explanatory view of the invalid information in the private application manager. When the check of the contents in the private application manager APCM1 and the check of the contents in the private license daemon APSM1 are unsuccessful, the
25 private application manager APCM1 sets an invalid information +ve (positive value) into the pipe. The application program AP1 reads the invalid information and terminates itself.

 Figure 24 is an explanatory view of the termination process in the private application manager. The private
30 application manager APCM1 terminates itself after sending of the termination instruction to the parent application program.

 Next, the abnormal termination of the private
35 license daemon is explained below.

 Figure 25 is an explanatory view of the restart in the private license daemon. When the private application

- 19 -

manager APCM1 detects the abnormal termination of the private license daemon APSM1, the private application manager APCM1 sends the request APRIR (Application Program Re-initiation Confirmation) to the license daemon LDP in order to restart the private license daemon APSM1.

Figure 26 is an explanatory view of the invalid notification from the license daemon. When the license daemon detects that the private license daemon APSM1 invalidates the request for a restart, the license daemon LDP sends the notification APRIC (Application Program Re-initiation Confirmation) indicating that the private license daemon APSM1 has not restarted, to the private application manager APCM1. Further, the private application manager APCM1 terminates itself, and the application program AP1 terminates.

Figure 27 is an explanatory view of the restart notification from the license daemon. When the license daemon LDP detects that the private license daemon APSM1 validates the request of restart, the license daemon LDP provides a new private license daemon APSM1' based on the fork instruction. The new private license daemon APSM1' updates the database, and sends the notification APRIC indicating that the private license daemon is restarted, to the private application manager APCM1.

Figure 28 is an explanatory view of the polling process from the private license daemon to the license daemon. The private license daemon APSM1 performs the periodical polling to the license daemon LDP. If the private license daemon APSM1 finds an EXIT of the license daemon LDP (step 1), the private license daemon APSM1 terminates itself (step 2). Accordingly, the normal operation of the application program AP1 is interrupted by the private application manager APCM1 (step 3), and the private application manager APCM1 loops trying to send APRIR to the LDP (step 4).

Figure 29 is an explanatory view of the termination

- 20 -

process in the license daemon. The license daemon LDP terminates in accordance with a request from a system administrator. There are two modes for termination, i.e., a normal termination (TR1) and another
5 termination (TR2). The license daemon LDP checks a valid application program AP1 having the license key after reception of the request of termination TR2, and the private license daemon APSM1 sends a termination instruction SDC (Shut Down Command) to the private
10 application manager APCM1 (step 1).

When the private application manager APCM1 receives the termination instruction SDC from the private license daemon APSM1, the private application manager APCM1 deals with this as a preferential message, and immediately
15 determines termination itself. The private application manager APCM1 forces the application program AP1 to surely send the instruction SIGUSR2 (one kind of signal in UNIX) to the private application program APCM1 before termination of the application program AP1 (step 2). The
20 private application program APCM1 performs the normal termination after reception of the instruction SIGUSR2 from the application program AP1 in accordance with the processes shown in Figs. 13 to 16.

Figure 30 is an explanatory view of the termination
25 process in the license daemon, when the private application manager APCM1 does not receive the instruction SIGUSR2 from the application program AP1 within a predetermined time in the process shown in Fig. 29. The private application manager APCM1 sends the
30 signal SIGKILL to the application program AP1 (step 1) so that the application program AP1 terminates (step 2).

The private application manager APCM1 terminates itself after a predetermined time (step 3), and notifies this termination to the license daemon LDP using the
35 instruction CORC (step 4). In this case, the license daemon LDP waits for the instruction CORC from the private application manager APCM1 during a predetermined

- 21 -

time. If the license daemon LDP does not receive the instruction CORC from the private application manager APCM1, the license daemon LDP terminates the private license daemon APSM1 (step 5).

5 Figure 31 is an explanatory view of the abnormal termination in the license daemon. When the license daemon LDP terminates abnormally, the private application manager APCM1 interrupts the normal operation in the application program AP1 (step 1), and interrupts itself
10 (step 2). When the license daemon LDP restarts after abnormal termination, the license daemon LDP checks whether the private license daemon APSM1 terminated after abnormal termination (step 3).

 The license daemon LDP checks the pending state of
15 the instruction APRIR in the private application manager APCM1. If the instruction APRIR is pending in the private application manager APCM1, the license daemon LDP receives the instruction APRIR from the private application manager APCM1 (step 4). The private
20 license daemon APSM1' recovers only when the license daemon LDP received the instruction APRIR from the private application manager APCM1 (step 5). The license daemon LDP starts normal operation.

 Figure 32 is an explanatory view of the abnormal
25 termination in the application program. When the application program AP1 terminates without sending of the instruction SIGUSR2 to the private application manager APCM1, the private application manager APCM1 sends the instruction APRR to the private license
30 daemon APSM1 (step 1), and checks existence of the application program AP1 (polling) (step 2). If the private application manager APCM1 finds that the application program AP1 terminated without notifying to the private application manager APCM1, the private
35 application manager APCM1 sends the instruction CORC to the license daemon LDP (step 3).

 Figure 33 is an explanatory view of the abnormal

- 22 -

termination of the private license daemon and the private application manager. After the private application manager APCM1 terminates (step 1), the application program AP1 also terminates after checking the information +ve (positive value) of the pipe provided between the application program AP1 and the private application manager APCM1 (step 2). The license daemon LDP receives the signal CORC from the private license daemon APSM1 (step 3), and releases the license key for the application program AP1.

Figure 34 is an explanatory view of the abnormal termination in both application program and the private application manager. When both the application program AP1 and the private application manager APCM1 terminated abnormally without notifying to either the license daemon LDP or the private license daemon APSM1 (step 1), the private license daemon APSM1 does not acquire the confirmation APPC (Application Program Poll Confirmation) for the request APPR (Application Program Poll Request) (step 2).

After the private license daemon APSM1 found the termination of the private application manager APCM1 (step 4), the private license daemon APSM1 checks for the existence of the application program AP1. If the application program AP1 exists, the private license daemon APSM1 waits for termination of the application program AP1 by polling thereto (step 5). After the above process, the private license daemon APSM1 notifies the termination of the application program AP1 to the license daemon LDP using the instruction CORC (step 6). The license daemon LDP releases the license key assigned to the application program AP1.

Figure 35 is an explanatory view of the abnormal termination in both application program and the private license daemon. When the application program AP1 terminates abnormally (step 1), the private application manager APCM1 detects this abnormal termination (step 2)

- 23 -

and sends the confirmation CORC to the license daemon LDP (step 3). The license daemon LDP releases all license keys which are held in the application program AP1 and terminates the private license daemon APSM1 (step 4). If
5 the private license daemon APSM1 has already terminated, there is no problem.

Figure 36 is an explanatory view of the abnormal termination in both license daemon and the private
license daemon. There are two cases in the termination
10 of the license daemon LDP and the private license daemon APSM1 as follows. 1) Both the license daemon LDP and the private license daemon APSM1 are killed by the signal SIGKILL. 2) The license daemon LDP is abnormally
killed, the private license daemon APSM1 checks the
15 license daemon LDP. When the private license daemon APSM1 found that the license daemon LDP terminated and executed the EXIT, the private license daemon APSM1 is killed by itself.

In this situation, the private application
20 manager APCM1 interrupts the application program AP1 after setting of information -ve (negative value) to the pipe between the application program AP1 and the private application manager APCM1 (step 1), and sends the request APRIR to the license daemon LDP (step 2).

Figure 37 is an explanatory view when the license
daemon LDP restarts after the situation described in the
explanation of Fig. 36. When the license daemon LDP
restarts, it recognizes the request APRIR (step 1), and
recovers a new private license daemon APSM1' in order to
30 communicate with the private application manager APCM1.

Figure 38 is an explanatory view of the system
structure. As shown in the drawing, for example, one
license daemon LDP can establish two groups, I and II,
each having the application program AP, the private
application manager APCM, and the private license
35 daemon APSM. According to this structure, it is possible to reduce the load of the license daemon LDP. As a

- 24 -

result, it is possible to eliminate the delay of the processing time in the software.

Next, the following explanations are given for the concrete decision method for the number of key according to the present invention.

Figure 39 is a flowchart for determining the number of license (number of key). Further, Figures 40(A) to 40(I) are actual function lists for determining the number of key, Figures 41(A) to 41(F) are explanatory views of the sample program, and Figures 42(A) to 42(C) are explanatory views of the execution result of the sample program.

The actual program using the functions shown in Figs. 40(A) to 40(I) is provided for processing a predetermined file. In this program, the following items, for example, file size to be processed, platform, number of CPU, capacity of memory, etc., are considered. Based on above consideration, this program has been designed so as to automatically start a plurality of processes in parallel so that it is possible to realize high speed processing of the program. Accordingly, it is necessary to increase/decrease the number of key in accordance with the number of processes to be started in parallel.

The actual function for determining the number of key is explained in detail below.

In Figs. 40(A) to 40(I), a first function "GetSystemParameters" gets five parameters, such as, number of CPU (Ncpu), page size of memory (Psize), number of page for all physical memory capacity (PhyPage), number of page of vacant memory capacity (AvPage), and platform (Platform). These parameters are delivered to a second function "DetermineNumberOfLicense".

The second function "DetermineNumberOfLicense" receives a size of the file and five parameters from the first function "GetSystemParameters", and determines the number of parallel processing. At the same time, the

- 25 -

second function "DetermineNumberOfLicense" determines the necessary number of keys and returns the number of keys as "return value".

At first, the first function "GetSystemParameters" is explained in detail regarding a first block (lines 53 to 68) and a second block (lines 71 to 89) in Fig. 40(D).

Regarding the first block (lines 53 to 68), the first function "GetSystemParameters" gets the following parameters, i.e., number of CPU (sysconf(_SC_NPPROCESSORS ONLN)), page size of memory (sysconf(_SC_PAGESIZE)), number of page of all physical memory capacity (sysconf(SC_PHYS_PAGES)), and number of page of vacant memory capacity (sysconf (_SC_AVPHYS__PAGES)). If there is an error in any one of parameters, the process is interrupted.

Regarding the second block (lines 71 to 89), the first function "GetSystemParameters" gets the platform (PLATFORM). If there is an error in the platform, the process is interrupted. When the first function "GetSystemParameters" gets normally all parameters, each value of the following parameters, i.e., number of CPU (Ncpu), page size of memory (Psize), number of page for all physical memory capacity (PhyPage), and number of page for vacant memory capacity (AvPage), are substituted for predetermined variables.

Second, the second function "DetermineNumberOfLicense" is explained in detail with reference to Fig. 39 and Figs. 40(F) to 40(I).

Regarding the first block (lines 125 to 134), whether a given argument is normal or not is checked. If the argument is abnormal, the process is interrupted. If the argument is normal, the number of license (number of key) is set to "default number" (NL=NL_DEFAULT) (see (1) on line 126).

Regarding the second block (lines 138 to 147), either if the number of CPU within the hardware is one, or if it is not a SPARC server, it is determined that the

- 26 -

parallel processing cannot be executed. In this case, it is determined that the number of process is one and the number of license is the default number. Then, the process is terminated in the range of this function. If
5 it is the SPARC server having two or more CPUs, the number of license (key) is set to double of default number ($NL = NL_DEFAULT + NL_DEFAULT$) (see (2) on line 139).

10 Regarding the third block (lines 150 to 157), a given file size is changed to the number of memory page (see (3) on line 150).

Regarding the fourth block (lines 160 to 181), whether the parallel processing (specified program) should be performed is determined, and the number of
15 parallel processes and the number of key are determined.

Regarding lines 163 to 166, if the value (file size * NL_MEM_FACTOR) is larger than the vacant memory capacity, it is meaningless to perform the parallel processing (According to an experiment, it is obvious
20 that the high speed processing cannot be obtained.). Accordingly, a single process is performed, and the number of key is determined to the default number (i.e., $NL = 2$).

Regarding lines 167 to 181, if the value (file size * NL_MEM_FACTOR) is smaller than the vacant memory capacity, the process is divided into the following three cases.
25

1) Regarding lines 167 to 171, if the value (file size * NL_MEM_FACTOR) is smaller than 10^6 bytes, it is
30 meaningless to perform the parallel processing (As mentioned above, as a result of an experiment, it is obvious that the high speed processing cannot be obtained.). Accordingly, a single process is performed, and the number of key is determined to (default number
35 * number of process)

2) Regarding lines 172 to 176, if the value (file

- 27 -

size * NL_MEM_FACTOR) is larger than 10^6 bytes, but less than $2 * 10^6$ bytes, two parallel processes are performed, and the number of key is determined by the value (default number * number of process).

5 3) Regarding lines 177 to 181, if the value (file size * NL_MEM_FACTOR) is larger than $2 * 10^6$ bytes, the number of parallel processing is performed based on the value (number of CPU - 1), the number of key is determined by the value (default number * number of
10 process).

Next, the explanation is given of one sample of the program shown in Figs. 41(A) to 41(F). This sample program utilizes the function "GetSystemParameters" and gets the information of the hardware, i.e., platform,
15 number of CPU, page size of memory, number of page for all physical memory capacity, and number of page of vacant memory capacity. Result of the above is displayed on a screen. Furthermore, by using the function "DetermineNumberOfLicense", the number of process and the
20 number of license (number of key) for twelve kinds of file sizes are displayed on the screen.

The sample program shown in Figs. 41(A) to 41(F) is explained in detail below.

Regarding the first block (lines 54 to 62), first,
25 variables are initialized (reset). That is, the number "0" is substituted for the information of the hardware, i.e., number of CPU(Ncpu), page size memory(Psize), number of page for all physical memory capacity(PhyPage), and number of page of vacant memory capacity(AvPage).
30 Further, "NULL" is substituted for the platform (PLATFORM).

Regarding the second block (lines 65 to 81), actual numbers of the hardware, i.e., number of CPU(Ncpu), page size of memory(Psize), number of page for all physical
35 memory capacity(PhyPage), and number of page of vacant memory capacity(AvPage), are determined and displayed on

- 28 -

the screen.

Regarding the third block (lines 85 to 103),
regarding twelve kinds of file sizes, the number of
parallel processing (number of process) and the number of
license (number of key) are determined based on the
5 information of the hardware given by the above processes.
The determined number of process and number of license
are displayed on the screen.

10 The result of execution of the sample program is
explained in detail with reference to Figs. 42(A) to
42(C). The sample program was executed by using two
kinds of workstations (model S-4/1 and model S-4/1000)
each using Solaris 2.5.

15 In Fig. 42(A), the sample program was executed by
the model S-4/1 as the platform. In this case, since the
number of CPU is only one, the number of license (number
of key) is always one regardless file size.

20 In Fig. 42(B), the sample program was executed by
the model S-4/1000 as the platform. In this case, the
SPARC server having four CPUs was used for this model,
and the number of parallel processing and the number of
license (number of key) are different each other in
accordance with the file size.

25 In Fig. 42(C), the sample program was also executed
by the model S-4/1000 as the platform. In this case, the
SPARC server having four CPUs was also used for this
model. Since the number of pages of the vacant memory
capacity (AvPage) is small, the parallel processing was
not performed. Accordingly, all number of license
30 (number of key) are two.

As is obvious from above explanations, basically,
the number of key decision function is created so as to
satisfy the following items.

35 First, the performance of the CPU has been checked
previously. When the CPU has high performance, it is
necessary to provide many number of keys.

Second, the vacant memory capacity and the file size

- 29 -

are compared each other. When the parallel processing should be performed, it is necessary to provide many number of keys.

5 Third, if there is another parameter to be considered, it is possible to change the number of license (number of key) by making functions similar to the above.

CAPABILITY OF UTILIZATION IN INDUSTRY

10 As explained above, according to the present invention, it is possible to solve various problems existing in both software maker and the user in the conventional license system. Accordingly, the software maker has always searched and developed an optimum license management in order to supply the software, which
15 have been developed at large expense, to the user based on a suitable price. As a result, the license management system according to the present invention can provide an issuance of license in which the sales strategy was sufficiently considered so that the present invention
20 includes very high possibility for utilization in an industry.

- 30 -

CLAIMS

1. A license management system for software which drives a single computer or a plurality of computers, comprising:

- 5 an application program for requesting a decision of the number of license which needs to drive itself and for receiving issuance of the license;
- a number of license decision means for determining the necessary number of license in accordance with the request from the application program; and
- 10 a license management means for issuing the number of license which was determined by the number of license decision means.

2. A license management system as claimed in claim 1, wherein the number of license decision means comprises means for determining the number of license based on the following multi-nominal function,

$$LK = f(x_1, x_2, \dots, x_n)$$

 where, LK is the number of license, x_1 to x_n are parameters which need to determine the number of license.

20

3. A license management system as claimed in claim 1, wherein the number of license decision means comprises means for acquiring necessary parameters to determine the number of license when the application program starts.

25

4. A license management system as claimed in claim 1, wherein the license management means comprises means for subtracting the necessary number of license from the number of licenses, which are held in the license management means, when the number of license is correct, and issuing the license to the application program when the number of license held therein are not negative.

30

5. A license management system as claimed in claim 1, wherein the number of license decision means is a daemon program.

35

- 31 -

6. A license management system as claimed in claim 1, further comprising a database for determining a multi-nominal function, wherein the multi-nominal function is changed in the database without changing the multi-nominal function in the number of license decision means.

7. A license management system for software which drives a single computer or a plurality of computers, an application program for requesting a decision of the number of license which needs to drive itself and for receiving issuance of the license;

a number of license decision function included in the application program for determining the necessary number of license in accordance with the request from the application program; and

a license management means for issuing the number of license which was determined by the number of license decision function.

8. A license management system as claimed in claim 7, wherein the number of license decision function is a function call.

9. A license management system as claimed in claim 7, wherein the number of license decision function comprises means for determining the number of license based on the following multi-nominal function,

$$LK = f(x_1, x_2, \dots, x_n)$$

where, LK is the number of license, x_1 to x_n are parameters which need to determine the number of license.

10. A license management system as claimed in claim 7, wherein the number of license decision function comprises means for acquiring values of the necessary parameters to determine the number of license when the application program starts.

11. A license management system as claimed in claim 7, wherein the license management means comprises means for subtracting the necessary number of license

- 32 -

from the number of licenses, which are held in the license management means, when the number of license is correct, and issuing the license to the application program when the number of license held therein are not negative.

12. A license management system as claimed in claim 7, further comprising a database for determining a multi-nominal function, wherein the multi-nominal function is changed in the database without changing the multi-nominal function in the number of license decision means.

13. A method for managing licenses in a license management system at least comprising a license management means, an application program and a number of license decision means, and for managing software which drives a single or a plurality of computers, the method comprising the steps of:

delivering parameters from the application program to the number of license decision means in order to substitute the number of license, which needed to start the application program itself, for the multi-nominal function, when the application program starts;

determining the number of license by checking values of the necessary parameters which need to determine the number of license in the number of license decision means, substituting the determined number of license for the parameters delivered from the application program, and returning the parameters to the application program;

notifying the necessary number of license from the application program to the license management means by delivering the parameters, and requesting the license; and

subtracting the necessary number of license from the number of licenses, which are held in the license management means, when the license management means receives a normal flag from the number of license

- 33 -

decision means, and returning the normal flag to the application program when the number of licenses, which are held in the license management means, is not negative, and issuing the license to the application program.

5 14. A method for managing licenses in a license management system as claimed in claim 13, wherein the license management means further comprises the step of
10 notifying the number of license notified from the application program to the number of license decision means in order to confirm whether the number of license requested by the application program is correct; and the
15 step of comparing the number of license notified from the application program to the license management means with the necessary number of license notified to the
 application program, and of returning the normal flag to the license management means when a result of comparison is correct.

20 15. A method for managing licenses in a license management system as claimed in claim 13, wherein the license management means further comprises the step of
 terminating the application program at a time when an abnormal flag is returned from the license management means.

25 16. A method for managing licenses in a license management system as claimed in claim 13, wherein the number of license decision means is a daemon program.

30 17. A method for managing licenses in a license management system at least comprising, an application program, a license management means, and a number of license decision function provided in the application program, and for managing software which drives a single or a plurality of computers, the method comprising the steps of;

35 delivering parameters from the application program to the number of license decision function in order to receive the necessary number of licenses which

- 34 -

needed to start itself when the application program starts;

determining the number of license by checking values of the necessary parameters which need to
5 determine the number of license in the number of license decision means, substituting the determined number of license for the parameters delivered from the application program, and returning the parameters to the application program;

10 notifying the necessary number of license from the application program to the license management means by delivering the parameters, and requesting the license; and

15 subtracting the necessary number of license from the number of licenses, which are held in the license management means, when the license management means receives a request from the application program, and returning a normal flag to the application program when the number of licenses, which are held in the
20 license management means, is not negative, and issuing the license to the application program.

18. A method for managing licenses in a license management system as claimed in claim 17, wherein the license management means further comprises the step of
25 terminating the application program at a time when an abnormal flag is returned from the license management means.

19. A method for managing licenses in a license management system as claimed in claim 17, wherein the
30 number of license decision function is a function call.

20. A method for managing licenses in a license management system at least comprising, a license management means, an application program, a number of license decision means, and an external database, and for
35 managing software which drives a single computer or a plurality of computers, the method comprising the steps of;

- 35 -

delivering parameters from the application program to the number of license decision means in order to substitute the necessary number of licenses which needed to start itself when the application program starts;

reading data from the database to the number of license decision means, determining a multi-nominal function and the number of license by checking the values of necessary parameters in order to determine the number of license, substituting the determined number of license for parameters delivered from the application program, and returning the parameters to the application program, in the number of license decision means;

notifying the necessary number of license from the application program to the license management means by delivering the parameters, and requesting the license; and

subtracting the necessary number of license from the number of licenses, which are held in the license management means, when the license management means receives a normal flag from the number of license decision means, returning the normal flag to the application program when the number of licenses, which are held in the license management means, is not negative, and issuing the license to the application program.

21. A method for managing licenses in a license management system as claimed in claim 20, wherein the license management means further comprises the step of notifying the number of license notified from the application program to the number of license decision means in order to confirm whether the number of license requested by the application program is correct; and the step of comparing the number of license notified from the application program to the license management means with the necessary number of license notified to the application program, and of returning the normal flag to

- 36 -

the license management means when a result of comparison is correct.

22. A method for managing licenses in a license management system as claimed in claim 20, wherein the
5 license management means further comprises the step of terminating the application program at a time when an abnormal flag is returned from the license management means.

23. A method for managing licenses in a license
10 management system as claimed in claim 20, wherein the number of license decision means is a daemon program.

24. A method for managing licenses in a license management system as claimed in claim 20, wherein the
15 database is changed when the multi-nominal function is changed without changing the number of license decision means.

25. A method for managing licenses in a license management system at least comprising, an application program, a license management means, a number of license
20 decision function provided in the application program and an external database, and for managing software which drives a single computer or a plurality of computers, the method comprising the steps of;

delivering parameters from the application
25 program to the number of license decision function in order to receive the necessary number of licenses which it needs to start itself when the application program starts;

reading data from the database to the number of
30 license decision means, determining a multi-nominal function and the number of license by checking the values of necessary parameters in order to determine the number of license, substituting the determined number of license for parameters delivered from the application program,
35 and returning the parameters to the application program, in the number of license decision means;

notifying the necessary number of license from

- 37 -

the application program to the license management means by delivering the parameters, and requesting the license; and

5 subtracting the necessary number of license from the number of licenses which are held in the license management means when the license management means receives a request from the application program, returning a normal flag to the application program when the number of licenses, which are held in the license
10 management means, is not negative, and issuing the license to the application program.

26. A method for managing licenses in a license management system as claimed in claim 25, wherein the license management means further comprises the step of
15 terminating the application program at a time when an abnormal flag is returned from the license management means.

27. A method for managing licenses in a license management system as claimed in claim 25, wherein the
20 number of license decision function is a function call.

28. A license management system as claimed in claim 1 or 7, further comprising a private license daemon separately provided from the license daemon of the license management means, and a private application
25 manager separately provided from the application program, wherein communication is performed between the private license daemon and the private application manager.

29. A license management system as claimed in claim 28, wherein a plurality sets of the private license
30 daemon and the private application manager are established for one license daemon.

30. A method of communicating information between a private license daemon and a private application manager, the private license daemon being separately provided from
35 a license daemon of a license management means, and the private application manager being separately provided from an application program, the method comprising the

- 38 -

steps of:

5 sending a request from the application program
to the license daemon in order to acquire an approval of
execution of the license when a user starts the
application program;

 generating a fork instruction in order to
establish the private license daemon; and

10 sending an approval of execution from the
private license daemon to the application program in
order to issue the license.

15 31. A method of communicating information as
claimed in claim 30, further comprising the step of
terminating the application program when the approval of
execution is not obtained from the private license
daemon.

20 32. A method of communicating information as
claimed in claim 30, further comprising the step of
notifying an invalidation from the application program to
the user when the approval of execution is obtained from
the private license daemon, but it includes an invalid
content.

25 33. A method of communicating information as
claimed in claim 30, further comprising the step of
generating the fork instruction in order to establish the
private application manager when the approval of
execution is obtained from the private license daemon,
and it includes a valid content; and of starting
communication between the private license daemon and the
private application manager.

30 34. A method of communicating information as
claimed in claim 30, further comprising the step of
sending a request of termination from the application
program to the private application manager when the user
terminates the application program.

35 35. A method of communicating information as
claimed in claim 30, further comprising the step of
setting an invalid information into a pipe from the

- 39 -

private application manager, and reading the invalid information of the pipe from the application program when the application program does not terminate after sending a request for the termination accidentally.

5 36. A method of communicating information as claimed in claim 30, further comprising the step of sending a termination report indicating termination of the application program from the private application manager to the license daemon when the application
10 program was terminated.

 37. A method of communicating information as claimed in claim 30, further comprising the step of writing the information from the license daemon to an external database when the application program was
15 terminated, and generating an instruction of termination from the license daemon to the private license daemon.

 38. A method of communicating information as claimed in claim 30, further comprising the step of periodically polling between the private license daemon
20 and the private application manager in order to check whether normal communication is performed therebetween.

 39. A method of communicating information as claimed in claim 38, further comprising the step of sending a check request from the private license daemon
25 to the private application manager in order to check whether the private application manager operates normally.

 40. A method of communicating information as claimed in claim 38, further comprising the step of
30 sending a normal notification from the private application manager to the private license daemon when check of contents from the private license daemon and check of contents in the private application manager
 itself are successful.

35 41. A method of communicating information as claimed in claim 38, further comprising the step of sending a termination instruction from the private

- 40 -

license daemon to the license daemon when the private license daemon gets an abnormal response from the private application manager, and terminating the private license daemon itself.

5 42. A method of communicating information as claimed in claim 38, further comprising the step of performing an invalidation process from the license daemon to the application program in order to release the license which was assigned from the license daemon to the
10 application program.

 43. A method of communicating information as claimed in claim 38, further comprising the step of sending a check request from the private application manager to the private license daemon in order to check
15 whether the private license daemon operates normally.

 44. A method of communicating information as claimed in claim 43, further comprising the step of sending a normal notification from the private license daemon to the private application manager when check of
20 contents from the private application manager and check of contents in the private license daemon itself are successful.

 45. A method of communicating information as claimed in claim 43, further comprising the step of
25 setting an invalid information into a pipe from the private application manager when check of contents in the private application manager itself and check of contents in the private license daemon are unsuccessful, and terminating the application program itself by reading the
30 invalid information from the pipe.

 46. A method of communicating information as claimed in claim 30, further comprising the step of sending a request of restart from the private application manager to the license daemon in order to restart the
35 private license daemon when the private application manager detects an abnormal termination of the private license daemon.

- 41 -

47. A method of communicating information as claimed in claim 46, further comprising the step of sending a notification indicating that the private license daemon is not restarted, to the private application manager when the license daemon detects that the private license daemon invalidates the request of restart.

48. A method of communicating information as claimed in claim 46, further comprising the step of providing a private license daemon based on a new fork from the license daemon when the license daemon detects that the private license daemon validates the request of restart and updating a database from the new private license daemon, and sending a notification indicating that the new private license daemon is restarted, from the new private license daemon to the private application manager.

49. A method of communicating information as claimed in claim 30, further comprising the step of periodically polling from the private license daemon to the license daemon.

Fig.1

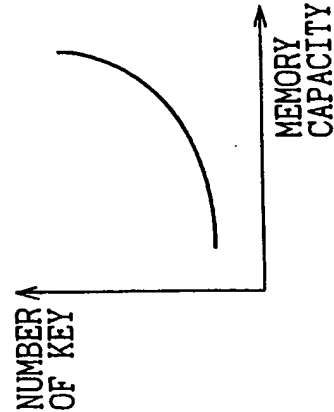


Fig.2



Fig.3

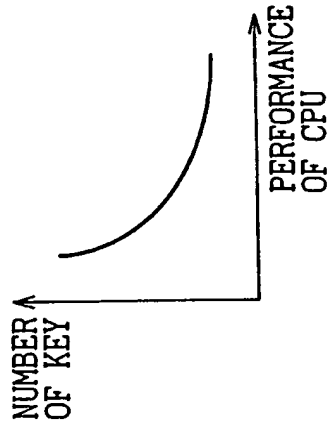


Fig.4

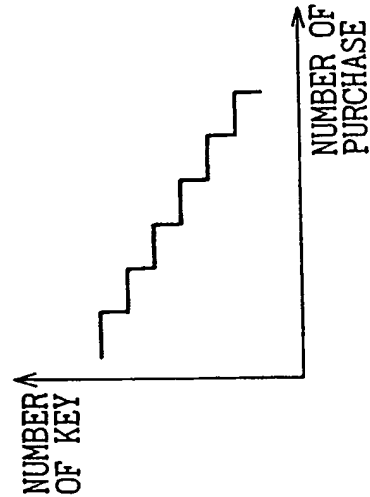
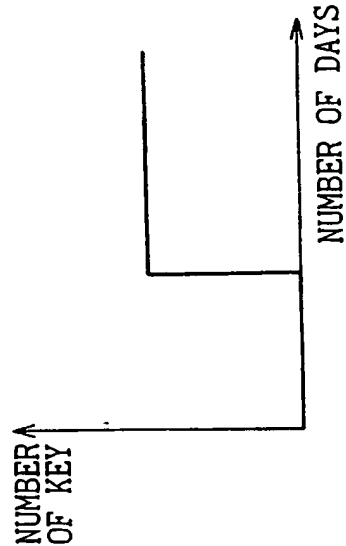


Fig.5



2/37

Fig.6

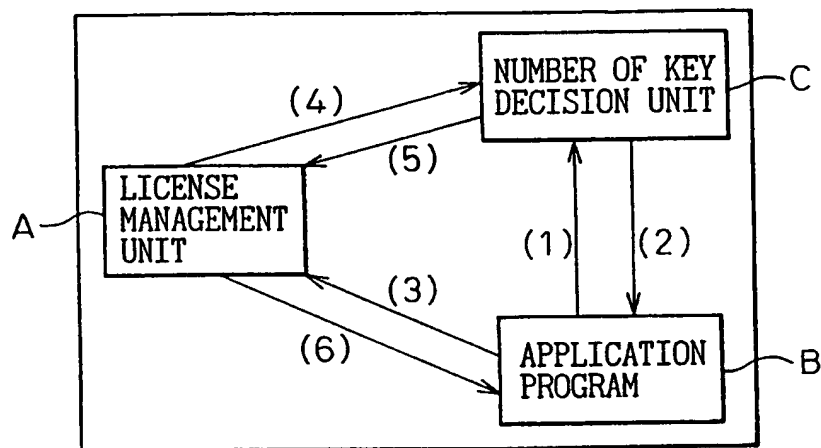
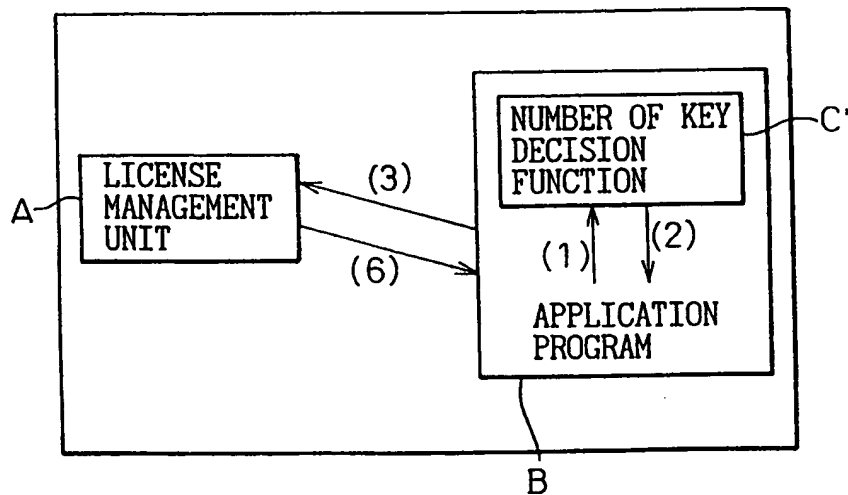


Fig.7



3/37

Fig.8

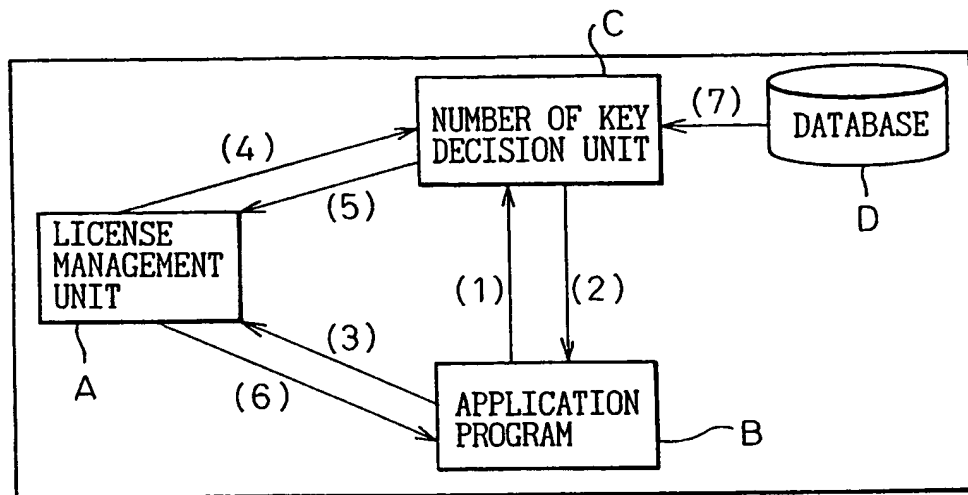
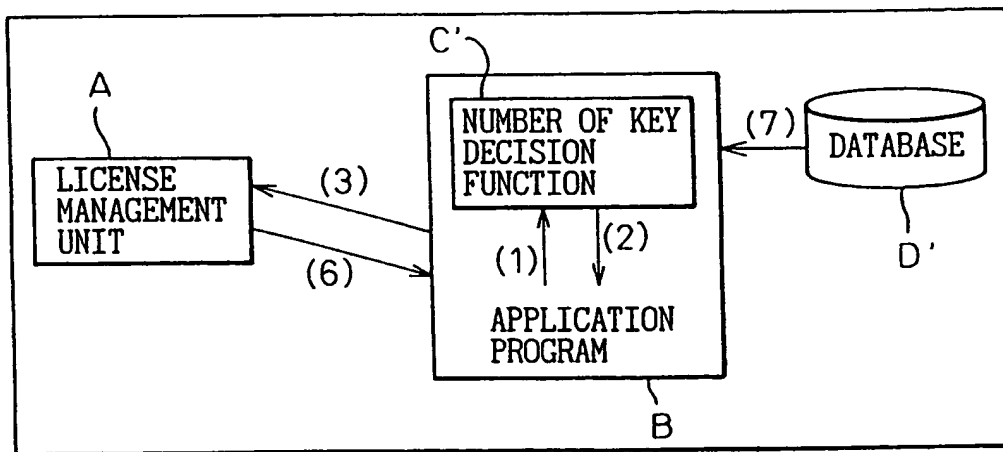


Fig.9



4/37

Fig.10



Fig.11

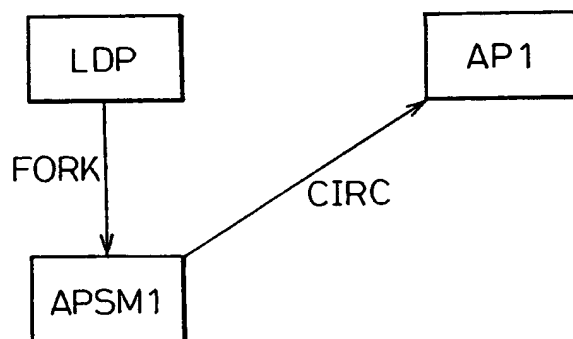
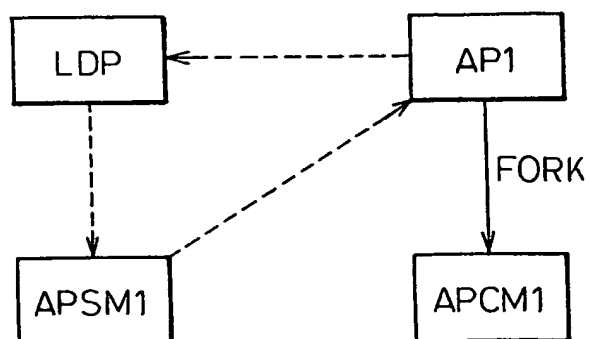


Fig.12



5/37

Fig.13

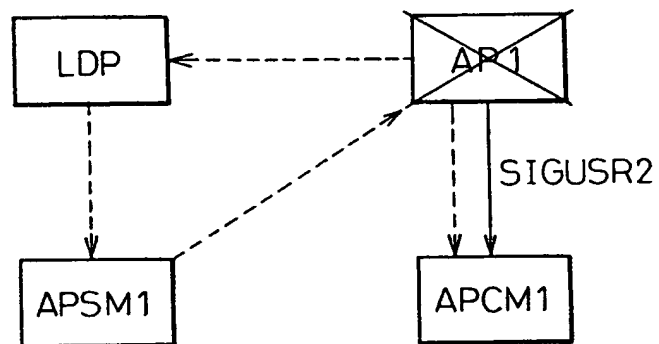
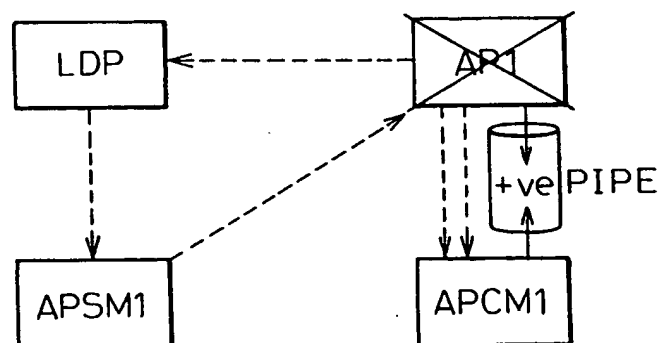


Fig.14



6/37

Fig.15

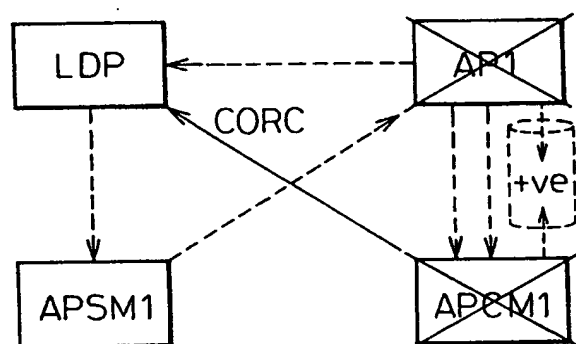
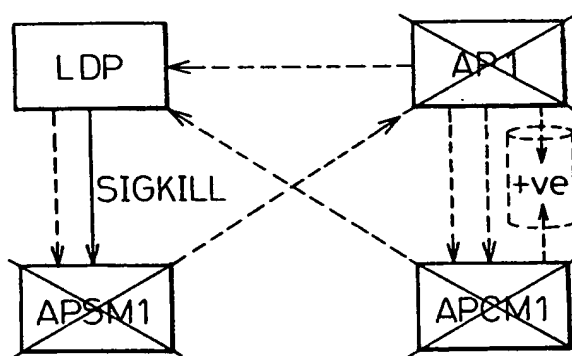


Fig.16



7/37

Fig.17

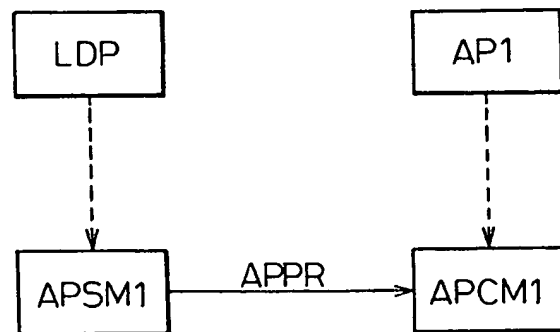
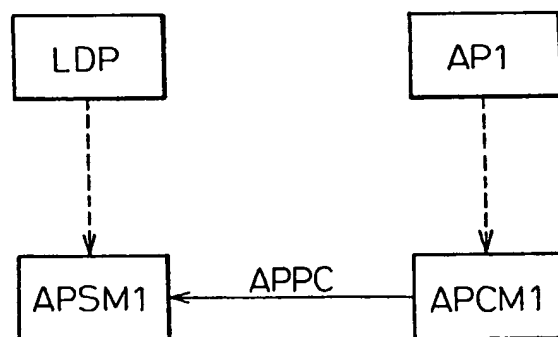


Fig.18



8/37

Fig.19

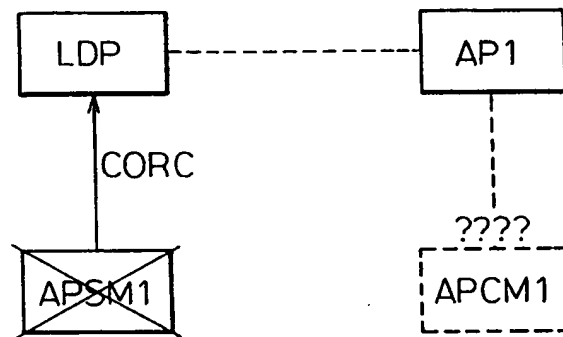
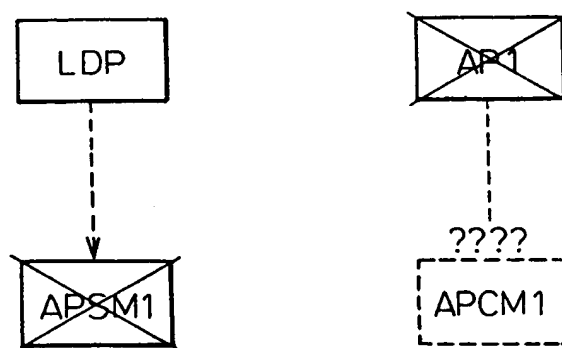


Fig.20



9/37

Fig.21

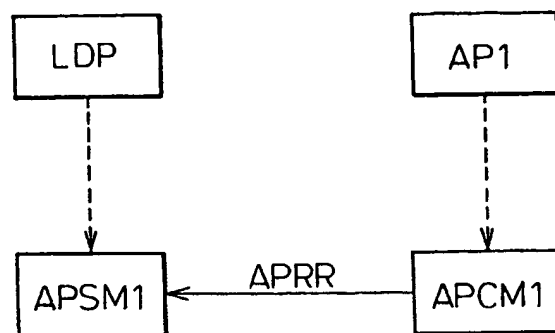
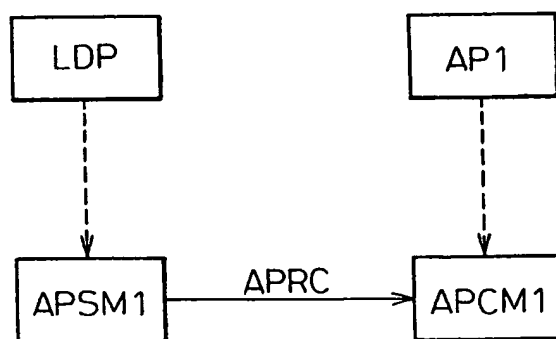


Fig.22



10/37

Fig.23

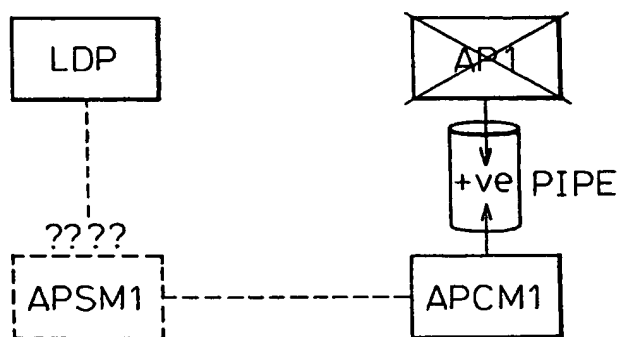
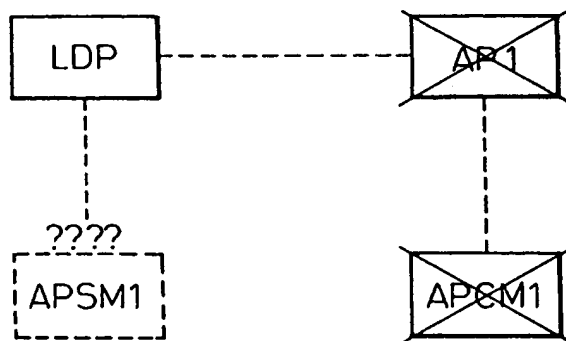


Fig.24



11/37

Fig.25

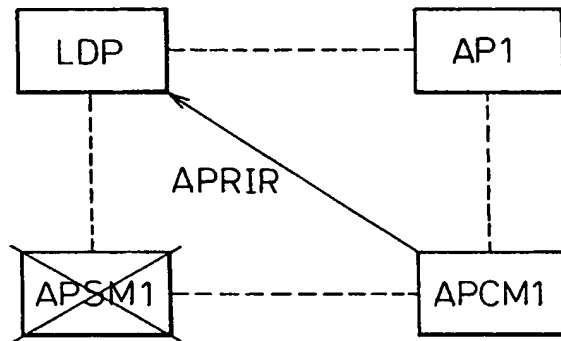
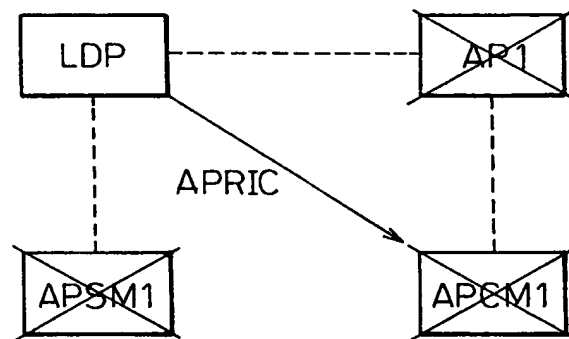


Fig.26



12/37

Fig.27

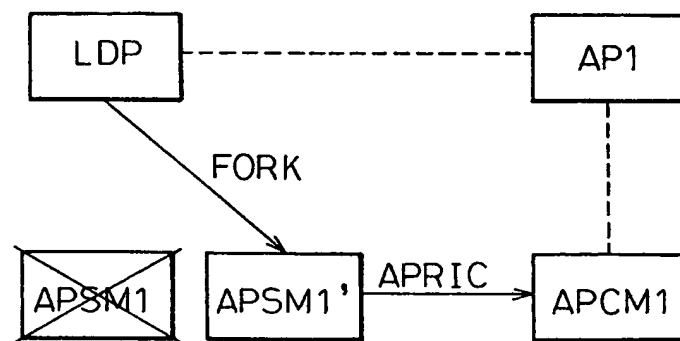
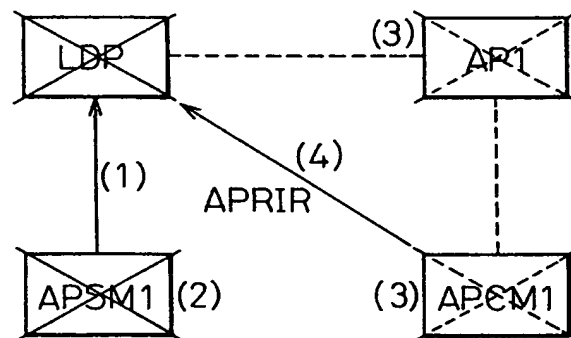


Fig.28



13/
37

Fig.29

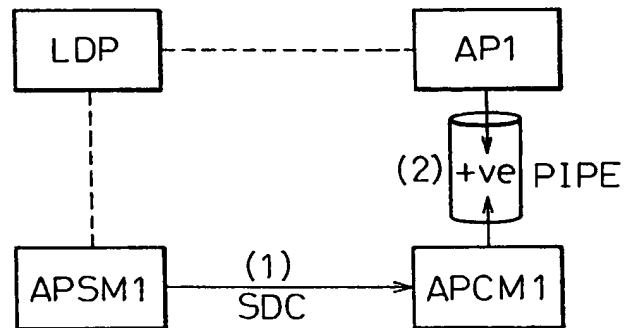
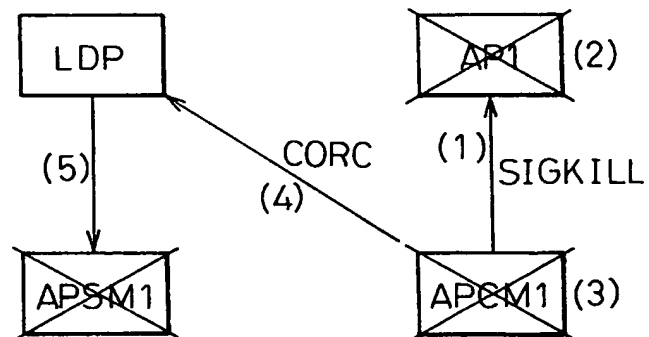


Fig.30



14/37

Fig.31

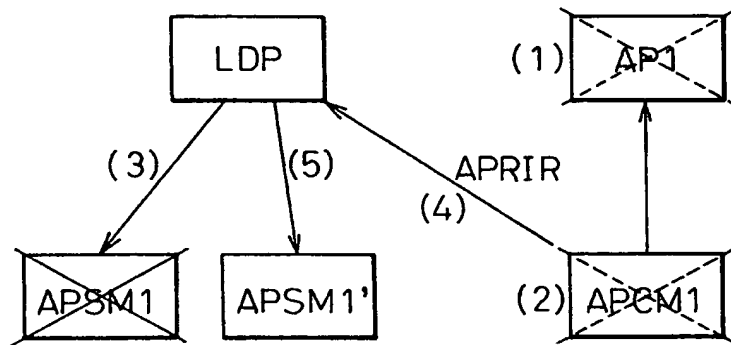
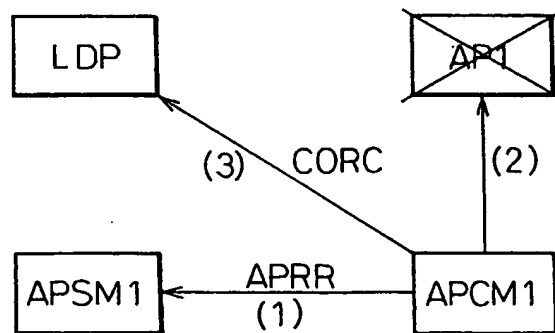


Fig.32



15/37

Fig.33

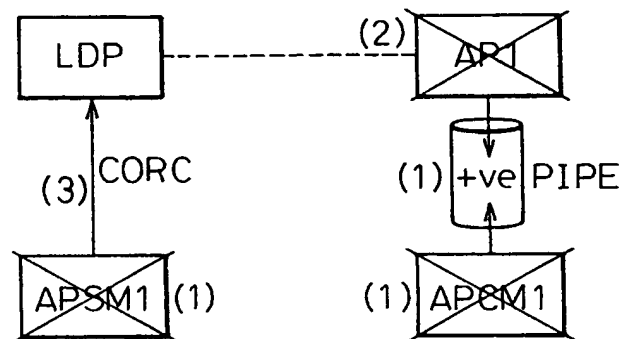
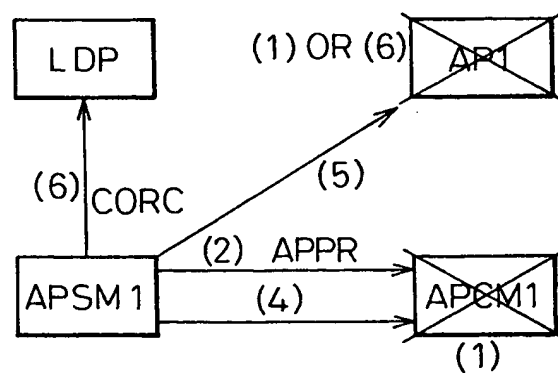


Fig.34



16/37

Fig.35

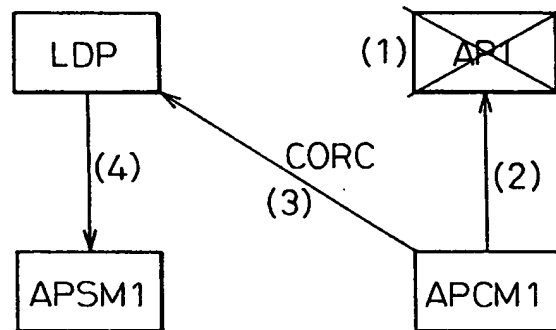
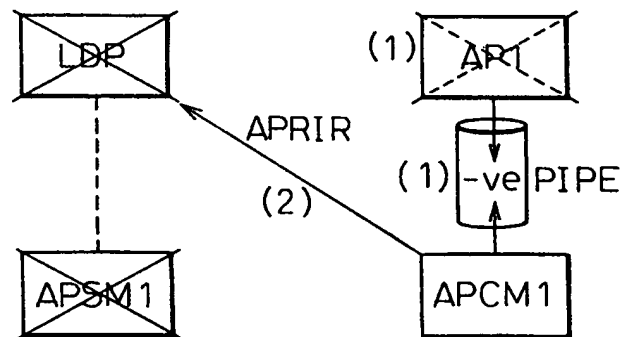


Fig.36



17/37

Fig.37

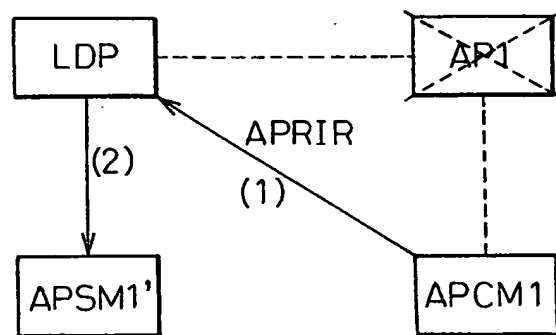
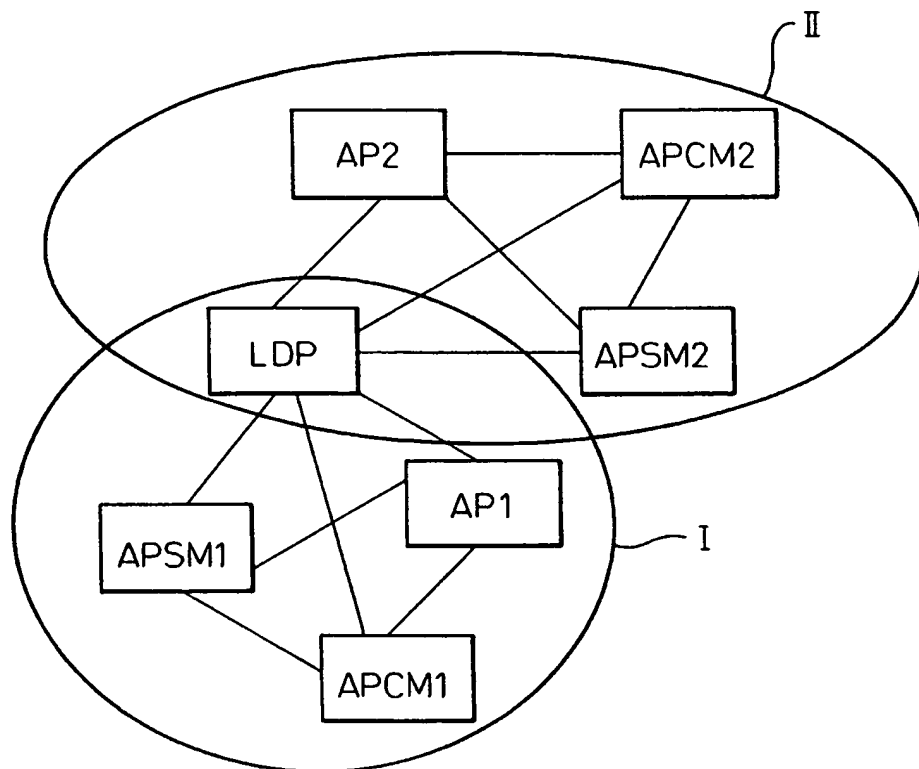
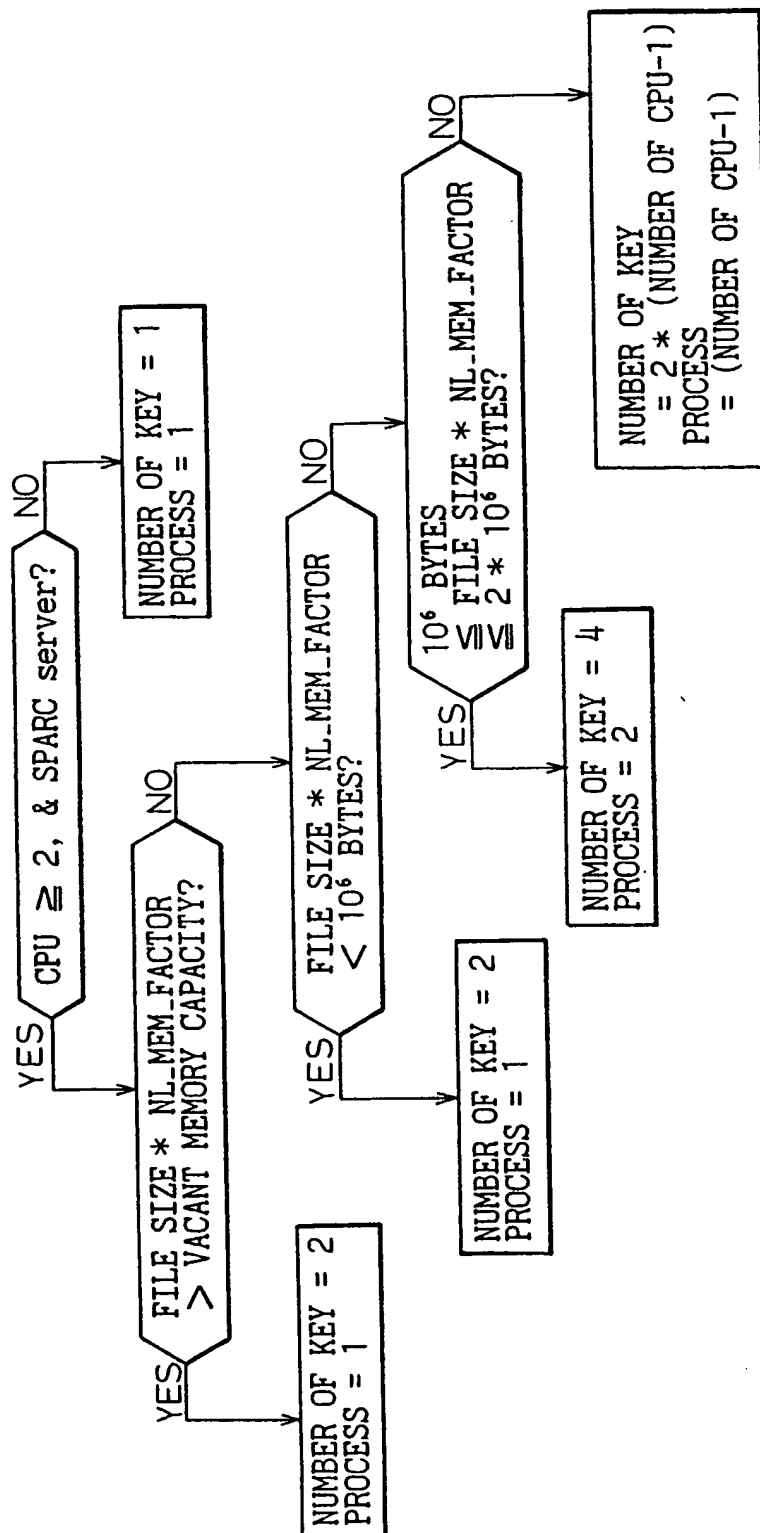


Fig.38



18/37

Fig. 39



19/37

Fig. 40(A)

```

1: /*****
2: /* Application: Product #MP439
3: /*-----*/
4: /* Source : license.c
5: /* Object : license.o
6: /*-----*/
7: /* (C) SHINKO ELECTRIC INDUSTRIES CO., LTD.
8: /* Production Engineering Division,
9: /* System Devices Development Department.
10: /*
11: /*****
12:
13: #include <stdio.h>
14: #include <string.h>
15: #include <unistd.h>
16: #include <sys/systeminfo.h>
17:
18:
19:
20: #define NL_DEFAULT 1
21: #define NL_MEM_FACTOR 1.8 /* determined experimentally */
22:

```

20/37

Fig. 40(B)

```

23: #define      FSIZE_500K      500000
24: #define      FSIZE_1M       1000000
25: #define      FSIZE_2M       2000000
26: #define      FSIZE_10M      10000000
27:
28:
29: /*-----*/
30: /* int  GetSystemParameters(  int *Ncpu, int *Psize, int *PhyPage,
31: /*                                     int *AvPage,  char **Platform )
32: /*
33: /* [R]      Ncpu:      number of CPUs on-line.
34: /* [R]      Psize:     System memory page size.
35: /* [R]      PhyPage:   Total number of pages of physical memory in system.
36: /* [R]      AvPage:    Number physical memory pages not currently
37: /*                                     in use by system.
38: /* [R]      Platform:  name of platform.
39: /*
40: /* # This function gets above mentioned 5 paramters.
41: /*
42: /* # Return = 0 on success.
43: /*          = -1 on failure.
44: /*-----*/

```

21/37

Fig. 40(C)

```

45: int GetSystemParameters( int *Ncpu, int *Psize, int *PhyPage,
46: int *AvPage, char **Platform )
47: {
48:     long    p1, p2, p3, p4;
49:     static char pname[64];
50:
51:
52:
53:     /*-----*/
54:     /* [1] get system limits */
55:     /*-----*/
56:     p1 = sysconf( _SC_NPROCESSORS_ONLN );
57:     p2 = sysconf( _SC_PAGESIZE );
58:     p3 = sysconf( _SC_PHYS_PAGES );
59:     p4 = sysconf( _SC_AVPHYS_PAGES );
60:
61:     if ( p1 == -1L || p2 == -1L || p3 == -1L || p4 == -1L ) {
62:         *Ncpu = 0;
63:         *Psize = 0;
64:         *PhyPage = 0;
65:         *AvPage = 0;
66:         *Platform = NULL;

```

22/37

Fig. 40(D)

```

67:         return(-1);
68:     }
69:
70:     /*-----*/
71:     /* [2] get platform name */
72:     /*-----*/
73:     if ( sysinfo( SI_PLATFORM, pname, sizeof(pname) ) <= 0 ) {
74:         *Ncpu    = 0;
75:         *Psize   = 0;
76:         *PhyPage = 0;
77:         *AvPage  = 0;
78:         *Platform = NULL;
79:         return(-1);
80:     }
81:     else {
82:         *Ncpu    = (int)p1;
83:         *Psize   = (int)p2;
84:         *PhyPage = (int)p3;
85:         *AvPage  = (int)p4;
86:         *Platform = pname;
87:         return(0);
88:     }

```

23/37

Fig. 40(E)

```

89:     }
90: }
91: }
92: /*-----*/
93: /* int DetermineNumberOfLicense(      long fsize,      int Psize,
94: /*                                int Ncpu,
95: /*                                int PhyPage,
96: /*                                int AvPage,      char *Platform,
97: /*                                int *Npp      )
98: /*
99: /* fsize:      size of file to be processed.
100: /* Ncpu:      number of CPUs on-line.
101: /* Psize:      System memory page size.
102: /* PhyPage:      Total number of pages of physical memory in system.
103: /* AvPage:      Number physical memory pages not currently
104: /*             in use by system.
105: /* Platform:      name of platform.
106: /* Npp:      number of parallel processing.
107: /*             1 : To perform single processing.
108: /*             n >= 2 : To perform parallel processing.
109: /*
110: /* # This function determines number of license required for

```

24/37

Fig. 40(F)

```

111: /* product #MP439. */
112: /* */
113: /* # Return = NN > 0 number of license to be obtained. */
114: /* = -1 on failure. */
115: /* ----- */
116: int DetermineNumberOfLicense( long fsize, int Psize, int PhyPage,
117: int Ncpu, int Psize, int PhyPage,
118: int AvPage, char *Platform,
119: int *Npp )
120: {
121:     int Fsize; /* file size page */
122:     int NL; /* number of license token */
123:
124:
125: /* ----- */
126: /* [1] check arguments and */
127: /* set default number of license required */
128: /* ----- */
129: if ( fsize <= 0L || Ncpu < 1 || Psize < 0 || PhyPage < 0 ||
130:     AvPage < 0 || PhyPage < AvPage || Platform == NULL ||
131:     Npp == NULL )
132:     return( -1 );

```

25/
37

Fig. 40(G)

```

133: else
134:     NL = NL_DEFAULT;
135:
136:
137:
138:     /*-----*/
139:     /* [2] check if the machine is SPARCserver having multi-CPUs */
140:     /*-----*/
141:     if ( Ncpu == 1 || strstr( Platform, "SPARCserver" ) == NULL ) {
142:         *Npp = 1;
143:         return( NL );
144:     }
145:     /**** if SPARCserver, double default value of NL ****/
146:     else
147:         NL = NL_DEFAULT + NL_DEFAULT;
148:
149:
150:     /*-----*/
151:     /* [3] get file size in page */
152:     /*-----*/
153:     Fsize = fsize % Psize;
154:     if ( Fsize == 0 )

```

26/37

Fig. 40(H)

```

155:         Fsize == fsize / Psize;
156:     else
157:         Fsize == fsize / Psize + 1;
158:
159:
160:     /*-----*/
161:     /* [4] check if enough unused memory pages exist */
162:     /*-----*/
163:     if ( Fsize * NL_MEM_FACTOR > AvPage ) {
164:         *Npp = 1;
165:         return( NL );
166:     }
167:     if ( fsize < FSIZE_1M ) {
168:         *Npp = 1;
169:         NL *= 1;      /* determine number of license */
170:         return( NL );
171:     }
172:     else if ( FSIZE_1M <= fsize && fsize <= FSIZE_2M ) {
173:         *Npp = 2;
174:         NL *= 2;      /* determine number of license */
175:         return( NL );
176:     }

```

Fig. 40(I)

```
177:      else {
178:          *Npp = Ncpu - 1;      /* set number of parallel processes */
179:          NL *= (Ncpu - 1);    /* determine number of license */
180:          return( NL );
181:      }
182:
183: }
184: /*-----*/
185:
```

28/
37

Fig. 41(A)

```

1:  /*****
2:  /* Application: sample to show how
3:  /*      GetSystemParameters() and
4:  /*      DetermineNumberOfLicense () work.
5:  /* -----
6:  /* Source : Sample.c
7:  /* Object : Sample.o
8:  /* -----
9:  /* (C) SHINKO ELECTRIC INDUSTRIES CO., LTD.
10: /*      Production Engineering Division,
11: /*      System Devices Development Department.
12: /*
13:  /*****
14:
15:  #include <stdio.h>
16:  #include <string.h>
17:  #include <stdlib.h>
18:
19:
20:  #define      NUM_FIZE      12
21:
22:

```

Fig. 41(B)

```

23: extern int GetSystemParameters( int *, int *, int *, int *, char ** );
24: extern int DetermineNumberOfLicense( long, int, int, int, char *, int *)
    ;
25:
26:
27: int main( int argc, char *argv [ ] )
28: {
29:     static long fsize[NUM_FIZE] = {
30:         10000,
31:         20000,
32:         50000,
33:         100000,
34:         200000,
35:         500000,
36:         1000000,
37:         2000000,
38:         5000000,
39:         10000000,
40:         20000000,
41:         50000000
42:     } ;

```

30/
37

Fig. 41(C)

```

43: int  ret;
44: int  cnt;
45:
46: int  Ncpu;
47: int  Psize;
48: int  PhyPage;
49: int  AvPage;
50: char *Platform;
51: int  Npp;
52:
53:
54: /*-----*/
55: /* [1] reset variables */
56: /*-----*/
57: Ncpu    = 0;
58: Psize   = 0;
59: PhyPage = 0;
60: AvPage  = 0;
61: Platform = NULL;
62: Npp     = 0;
63:
64:

```

Fig. 41(D)

```

65:  /*-----*/
66:  /* [2] get system parameters */
67:  /*-----*/
68:  ret = GetSystemParameters( &Ncpu, &Psize, &PhyPage, &AvPage, &Platform
);
69:  if (ret != 0) {
70:      (void)fprintf( stderr, "! Error : GetSystemParameters() !\n");
71:      exit(1);
72:  }
73:  else {
74:      (void)printf( "-----\n" );
75:      (void)printf( "(1) Platform      =%s\n", Platform );
76:      (void)printf( "(2) Number of CPU =%d\n", Ncpu );
77:      (void)printf( "(3) Page size   =%d\n", Psize );
78:      (void)printf( "(4) PhyPage    =%d\n", PhyPage );
79:      (void)printf( "(5) AvPage     =%d\n", AvPage );
80:      (void)printf( "-----\n" );
81:  }
82:

```

Fig. 41(E)

```

83:
84:
85:  /*-----*/
86:  /* [3] determine number of license for each file size */
87:  /*-----*/
88:  for (cnt = 0; cnt < NUM_FIZE; cnt++) {
89:
90:      ret = DetermineNumberOfLicense( fsize[cnt], Ncpu, Psize, PhyPage,
      AvPage,
      Platform, &Npp);
91:
92:
93:      if (ret == -1) {
94:          (void)fprintf( stderr, "! Error : DetermineNumberOfLicense() !
          \n" );
95:          exit(1);
96:      }
97:      else {
98:          (void)printf(
99:              "File size = % 81d  Number of Process = % d  Number of license
              = % d\n",
              p, ret );
100:
              fsize[cnt], Np

```

33/37

Fig. 41(F)

```
101:      }  
102:  } /* for-ent */  
103:  (void)printf("\n");  
104:  return(0);  
105:  }  
106:  }  
107:  }  
108:  }  
109: }
```

34/37

Fig. 42(A)

kai13 {sato} (8)% sample				
(1) Platform	= SUNW,Sun_4_65			
(2) Number of CPU	= 1			
(3) Page size	= 4096			
(4) PhyPage	= 10240			
(5) AvPage	= 129			
File size =	10000	Number of Process =	1	Number of License = 1
File size =	20000	Number of Process =	1	Number of License = 1
File size =	50000	Number of Process =	1	Number of License = 1
File size =	100000	Number of Process =	1	Number of License = 1
File size =	200000	Number of Process =	1	Number of License = 1
File size =	500000	Number of Process =	1	Number of License = 1
File size =	1000000	Number of Process =	1	Number of License = 1
File size =	2000000	Number of Process =	1	Number of License = 1
File size =	5000000	Number of Process =	1	Number of License = 1
File size =	10000000	Number of Process =	1	Number of License = 1
File size =	20000000	Number of Process =	1	Number of License = 1
File size =	50000000	Number of Process =	1	Number of License = 1
kai13 {sato} (9)%				

Fig.42(B)

kai12 {sato} (13)% sample				
(1) Platform	= SUNW,SPARCserver-1000			
(2) Number of CPU	= 4			
(3) Page size	= 4096			
(4) PhyPage	= 32768			
(5) AvPage	= 8272			
File size = 10000	Number of Process = 1	Number of License = 2		
File size = 20000	Number of Process = 1	Number of License = 2		
File size = 50000	Number of Process = 1	Number of License = 2		
File size = 100000	Number of Process = 1	Number of License = 2		
File size = 200000	Number of Process = 1	Number of License = 2		
File size = 500000	Number of Process = 1	Number of License = 2		
File size = 1000000	Number of Process = 2	Number of License = 4		
File size = 2000000	Number of Process = 2	Number of License = 4		
File size = 5000000	Number of Process = 3	Number of License = 6		
File size = 10000000	Number of Process = 3	Number of License = 6		
File size = 20000000	Number of Process = 1	Number of License = 2		
File size = 50000000	Number of Process = 1	Number of License = 2		
kai12 {sato} (14)%				

36/37

Fig. 42(C)

kai12 {sato} (6)% sample

(1) Platform = SUNW, SPARCserver-1000
 (2) Number of CPU = 4
 (3) Page size = 4096
 (4) PhyPage = 32768
 (5) AvPage = 259

File size =	10000	Number of Process =	1	Number of License =	2
File size =	20000	Number of Process =	1	Number of License =	2
File size =	50000	Number of Process =	1	Number of License =	2
File size =	100000	Number of Process =	1	Number of License =	2
File size =	200000	Number of Process =	1	Number of License =	2
File size =	500000	Number of Process =	1	Number of License =	2
File size =	1000000	Number of Process =	1	Number of License =	2
File size =	2000000	Number of Process =	1	Number of License =	2
File size =	5000000	Number of Process =	1	Number of License =	2
File size =	10000000	Number of Process =	1	Number of License =	2
File size =	20000000	Number of Process =	1	Number of License =	2
File size =	50000000	Number of Process =	1	Number of License =	2

kai12 {sato} (7)%

37/37

A LIST OF REFERENCE NUMBERS

- A: LICENSE MANAGEMENT UNIT
- B: APPLICATION PROGRAM
- C: NUMBER OF KEY DECISION UNIT
- C': NUMBER OF KEY DECISION FUNCTION
- D: DATABASE
- D': DATABASE

INTERNATIONAL SEARCH REPORT

International Application No

PCT/JP 97/02460

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/60

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 260 999 A (WYMAN ROBERT M) 9 November 1993 see column 6, line 41 - column 8, line 40	1-49
A	US 5 375 206 A (HUNTER JAMES D ET AL) 20 December 1994 see abstract	1,7,13, 17,20, 25,30
A	WO 93 01550 A (INFOLOGIC SOFTWARE INC) 21 January 1993 see abstract	1,7,13, 17,20, 25,30

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

- "A" document defining the general state of the art which is not considered to be of particular relevance
- "E" earlier document but published on or after the international filing date
- "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- "O" document referring to an oral disclosure, use, exhibition or other means
- "P" document published prior to the international filing date but later than the priority date claimed

- "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- "&" document member of the same patent family

Date of the actual completion of the international search

9 March 1998

Date of mailing of the international search report

13/03/1998

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax (+31-70) 340-3016

Authorized officer

Suendermann, R

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/JP 97/02460

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5260999 A	09-11-93	AU 659652 B	25-05-95
		AU 2015892 A	21-12-92
		EP 0538453 A	28-04-93
		IL 102114 A	14-05-96
		NZ 243277 A	26-10-95
		WO 9220022 A	12-11-92
US 5375206 A	20-12-94	JP 5274275 A	22-10-93
WO 9301550 A	21-01-93	AU 2305292 A	11-02-93